Contents

A. Illustrative Comparison of Optimizations

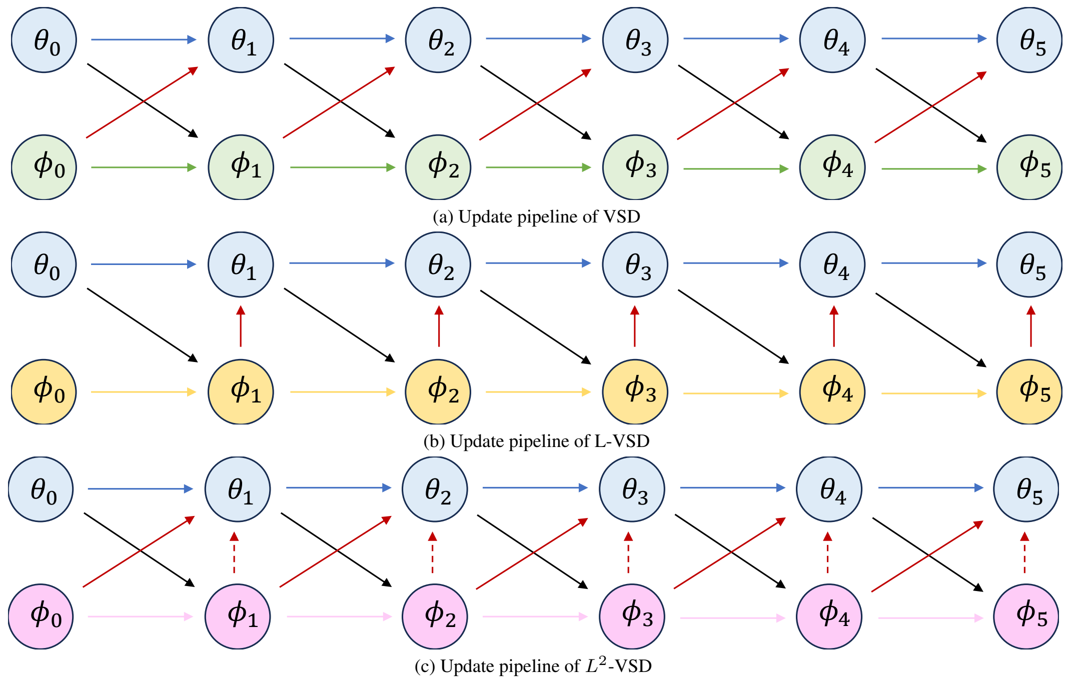


Figure 1. **Overview of VSD, L-VSD and L^2-VSD training.**

We present an illustrative overview about the updating pipeline of VSD, L-VSD and L^2-VSD respectively. As stated in section 2.2 of main paper, we use θ_i and ϕ_i to represent the 3D and the LoRA models at i_{th} iteration respectively. We use arrows with different colors to represent state transition dependency. We argue that red dashed arrow pointing from ϕ_i to θ_i is important for better results' quality.

More illustrative 2D gaussian examples. To gain a more complete view about the convergence of VSD, we conduct two additional gaussian experiments as shown in Fig. 2 and Fig. 3. In the example of Fig.2 of main paper, we only sample one point to keep as the same in ProlificDreamer, in which only one view of 3D object is rendered. In Fig. 2, we increase the number to 4, finding that the error introduced by optimization order could be mitigated to some extent. This evidence enlightens us that VSD with multi-view estimation may perform better, part of which has been proved in MVDream [19]. Besides, we also show the bad convergence if we overfit LoRA model on current sampled views in Fig. 3. It's worth noting that the distribution tends to lie between the intersection of two gaussian modals, making the views more saturated, which is coherent to the finding in section 3.3 of main paper. We provide the reproducible example code in Appendix. F.

B. Experiment Implementation

B.1. Main Experiments Details

Qualitative Results. In this section, we provide more details on the implementation of L^2-VSD and the compared baseline methods. All of them are implemented under the threestudio framework directly in the first stage coarse generation, without geometry refinement and texture refinement, following [29]. For the coarse generation stage, we adopt foreground-background disentangled hash-encoded NeRF [11] as the underlying 3D representation. All scenes are trained for 15k steps for the coarse stage, in case of geometry or texture collapse. At each interation, we randomly render one view. Different from classic settings, we adjust the rendering resolution directly as 64 $\times$ 64 in the low resolution experiments. And increase to 256 $\times$ 256 resolution in the high resolution experiments. All of our experiments are conducted on a single NVIDIA GeForce RTX 3090.

Quantitative Results. To compute FID [2], we sample N images using pretrained latent diffusion model given text prompts as the ground truth image dataset, and render N views uniformly distributed over a unit sphere from the optimized 3D scene as the generated image dataset. Then standard FID is computed between these two sets of images. To compute CLIP similarity, we render 120 views from the generated 3D representations, and for each view, we obtain an embedding

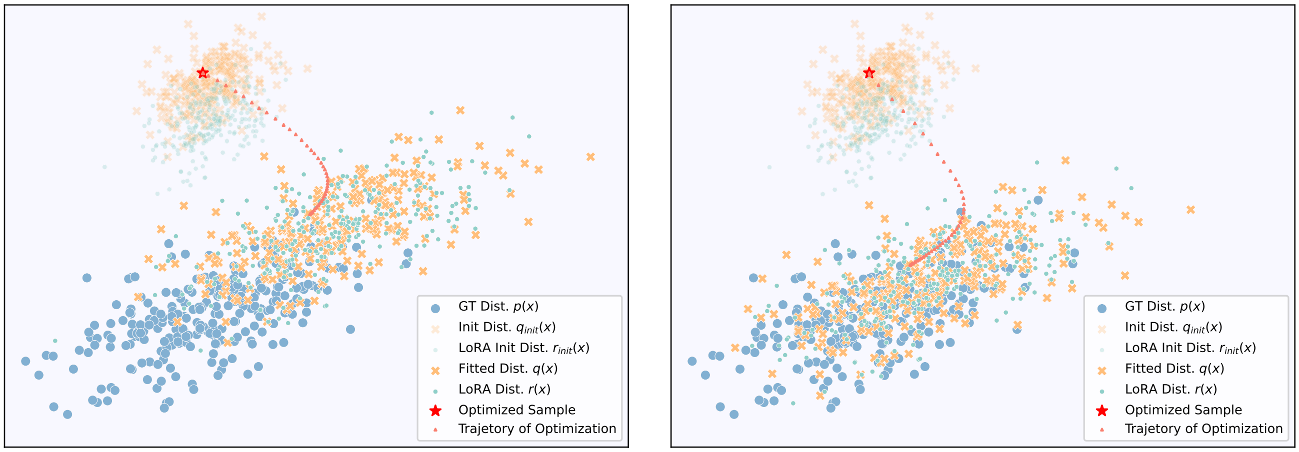


Figure 2. **Comparison of VSD and L-VSD with more render samples.** In this example, we sample 4 points in each iteration.

,

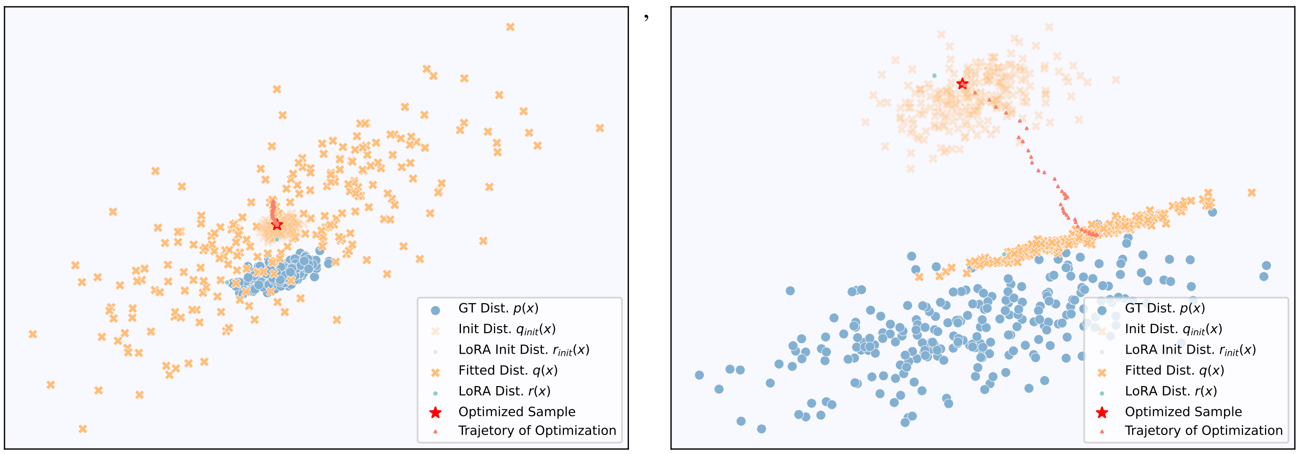


Figure 3. **Exploring the impact of $r(x)$ overfitting on rendered samples.** In this example, $r(x)$ is delta distribution as we overfit it on x in each iteration.

vector and text embedding vector through the image and text encoder of a CLIP model. We use the CLIP ViT-B/16 model [13].

B.2. High Order $\Delta\epsilon_{high}$omputation Details

As mentioned above in section 4.1 of main paper, we can comppute $\Delta\epsilon_{high}$ as $\epsilon_{\phi_{i+1}}(x_t, t, c, y) - \epsilon_{\phi_i}(x_t, t, c, y) - \Delta\epsilon_{first}$. In practice, we implement this computation during the training process of L-VSD. We copy an additional LoRA model to restore the LoRA parameters before being updated. Then in each optimization iteration for θ_i, the LoRA model performs forward passes for three times to calculate the ϵ_{ϕ_i}, $\epsilon_{\phi_{i+1}}$ and $\Delta\epsilon_{first}$ respectively.

B.3. Computation Costs

While our method can take 0.3s per iteration than baseline, our method can converge much faster as demonstrated by Fig.2 of main paper. Usually our method can produce satifying results in 10k steps, while VSD needs 15k steps or more. As a result, our method performs slightly more efficient than VSD, with higher quality. Even more, with last-layer approximation, we can achieve a trade-off between efficiency and performance.

	Time cost (s/iteration)	Converge Steps	Total time(hrs)
VSD	~ 0.7	$\sim$15k	~ 3
L^2-VSD	~ 1.0	$\sim$10k	~ 2.7
L^2-VSD (last-layer)	~ 0.8	$\sim$11k	~ 2.5

Table 1. **Computation efficiency.** We present the time cost in each iteration. We measure the average time on the threestudio framework.

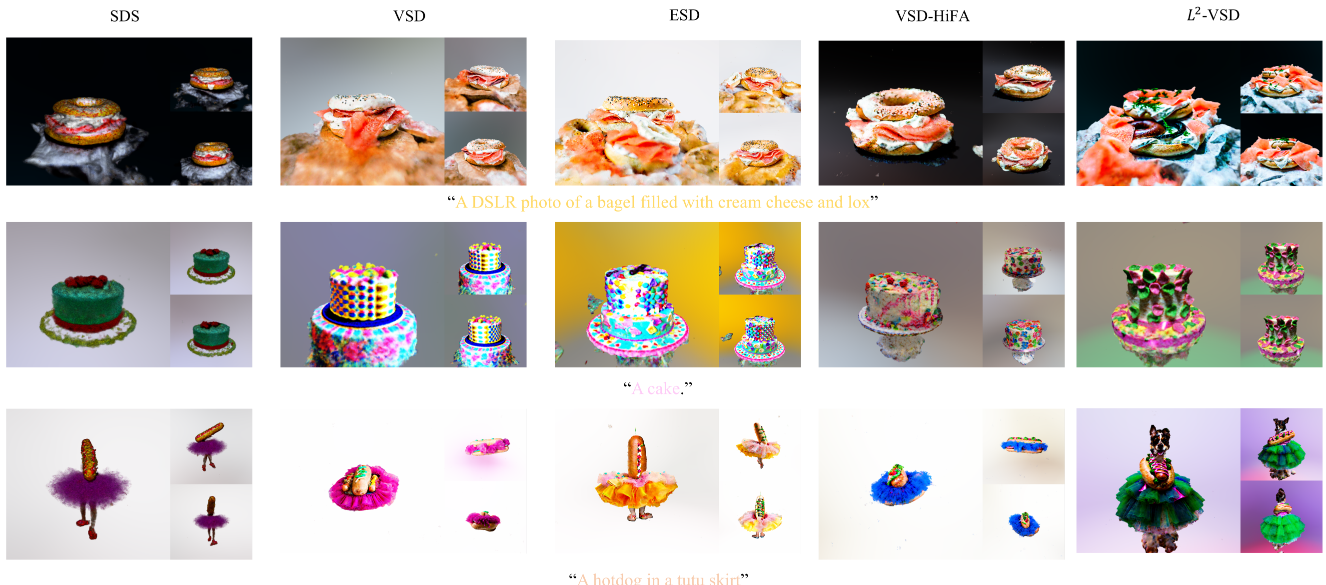


Figure 4. **Qualitative comparison with low resolution of 64**. L^2-VSD can generate highly detailed 3D assets even with low resolution, while the other baselines (except for HiFA), suffering from geometry-texture co-training, tend to be blurry and have floaters.

C. More Experiment Results

C.1. Failure Cases Produced by L-VSD

Figure 5. **Visualization of Failure Process.** The upper row result is generated with original learning rate while the lower one is generated with scaling the learning rate by 0.1. Each row corresponds to a continue optimization process. Our prompt is "an astronaut riding a horse".

We show an example of failure case produced by L-VSD in Fig. 5. We can observe that the upper one becomes over-saturated faster than the below one. Though the below one collapses much slower, it can't converge to a realistic case. Also, we provide all the L-VSD results in Fig. 6, which reflects the unstable generation quality by naive L-VSD.

C.2. Generalization on other representations

We provide the results generated in the second "geometry refinement" and third "texture refinement" stage in Fig. 7 and Fig. 8. In Fig. 7, the 3D objects are initialized with the results in the first stage. While in Fig. 8, we control the geometry initialization to be the same for our method and VSD, thus directly comparing the texture generation quality. In Fig. 8, VSD

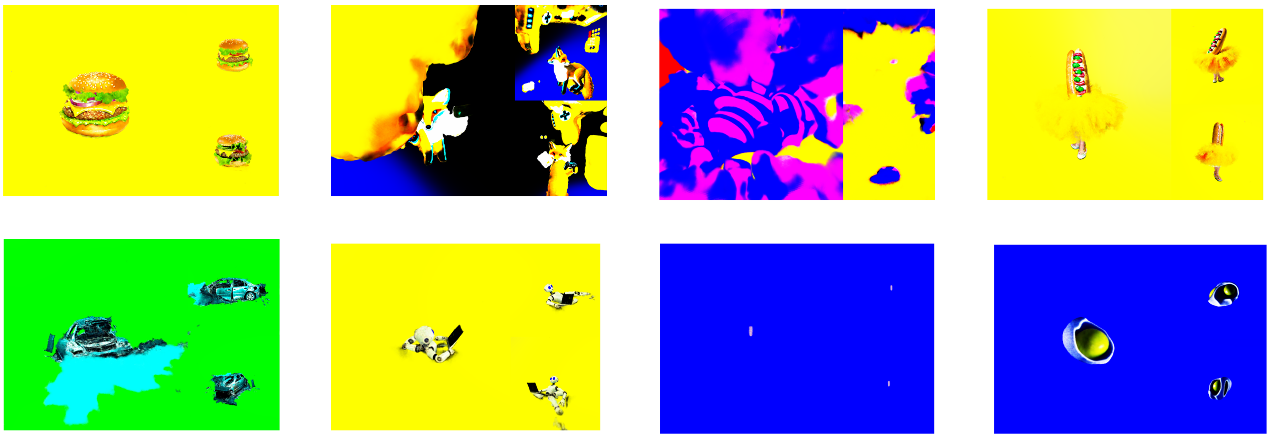


Figure 6. **Results of L-VSD.** These results are generated with the same prompts in Fig.1 and Fig.6 of main paper. As we can observe, naive L-VSD usually fails in generating realistic objects, which is supported by our Gaussian example in Fig.2 of main paper.

generates destroyed car with random red color, connecting destroyed car with a fire but our method generates more purely. And the texture of hand and the bowl in the bottom is also more realistic. As these two stages represent in mesh, we believe this comparison reflects the generalization of our method on other representations.

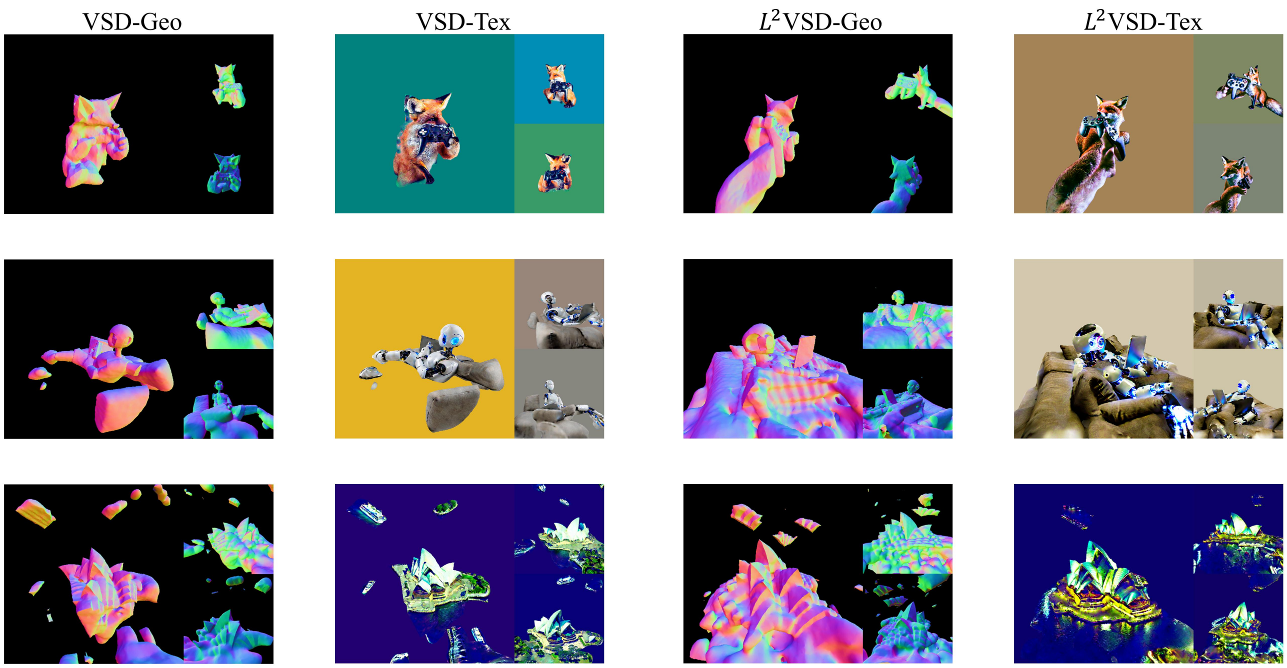


Figure 7. **Comparison at second and third stages**. We initial the objects with first-stage's results and compare the geometry and texture refinement. As shown in the figure, the geometry generated by our method is more complete and texture generated by our method is much more realistic.

C.3. Loss curve comparison at initial stages

To have a better understanding of the optimization behavior in section 3.1 of main paper, we show the loss curve at initial stage in Fig. 9a. As shown by the curve, the loss is in similar level at the start of distillation, which is probably because the

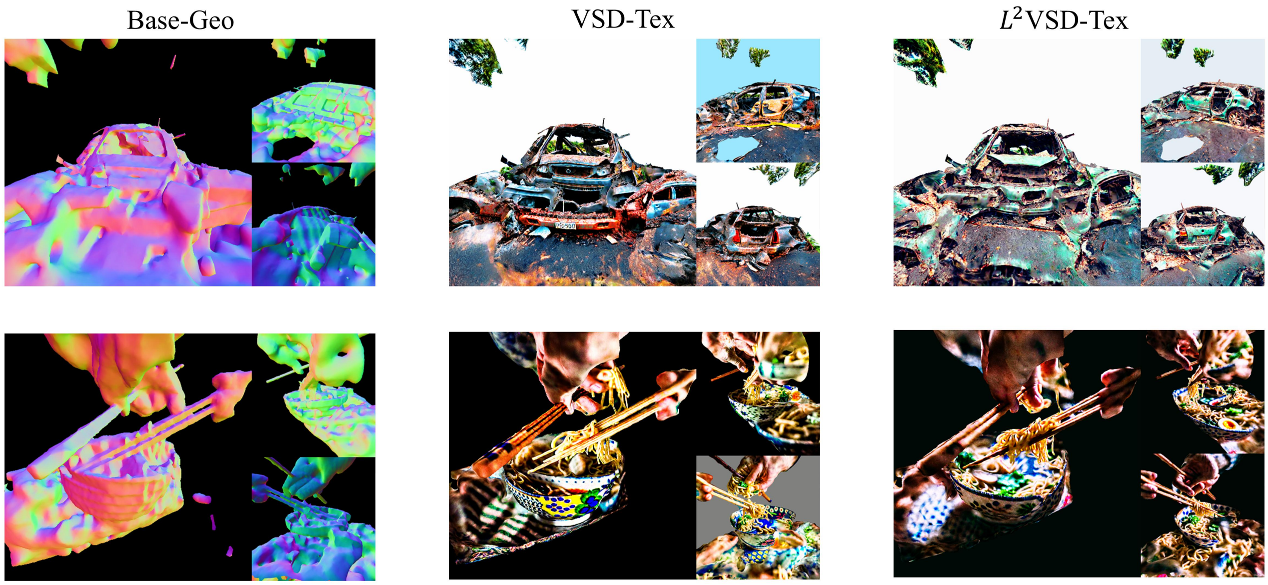
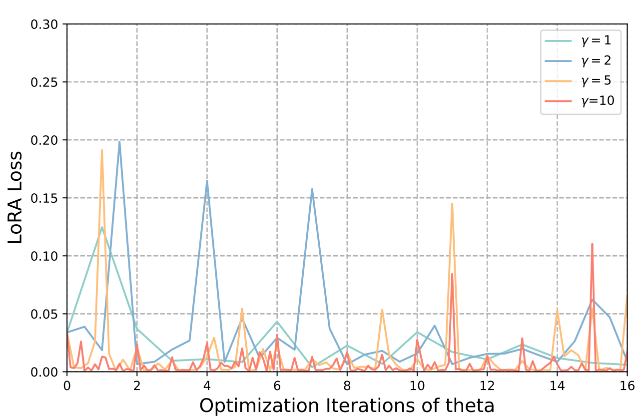
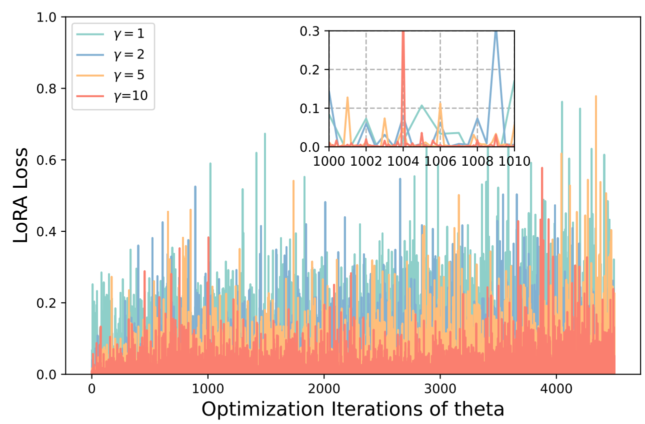


Figure 8. **Comparison on texture representation**. We use VSD and our method to generate texture conditioned on the same geometry initialization. Prompts: (Upper)"a completely destroyed car" ;(Bottom)"a zoomed out DSLR photo of a pair of floating chopsticks picking up noodles out of a bowl of ramen".



(a) **VSD multi-LoRA initial loss**: At the start of distillation, the loss with different LoRA steps is in the similar level.



(b) **Multi samples averaged loss curve.** We average the LoRA loss on 3 samples, finding the general pattern of loss variation.

Figure 9. **More Loss Curve.**

,

objects don't form into clear shape yet. So the predicted noises are all likely to be gaussian.

Also, to verify the generality of this phenomenon, we test on multiple samples and measure the average LoRA loss to provide more convincing results, which is shown in Fig. 9b. The conclusion holds as the same as in the section 3.1 of main paper. Also, we provide one sample "crown" other than "hamburger" to augment the proof.

C.4. Ablation of Generation with high-order term

We provide the results of one important ablation experiment in Fig. 11. We compare the results produced by VSD, L^2-VSD and HL-VSD(high-order lookahead VSD). In HL-VSD, we use the high-order term instead of the linear term to correct the score. As shown in the figure, the results all collapse and become irrecognizable, which proves the effectiveness and necessity of linearied lookahead.

Figure 10. **VSD LoRA Comparison**.

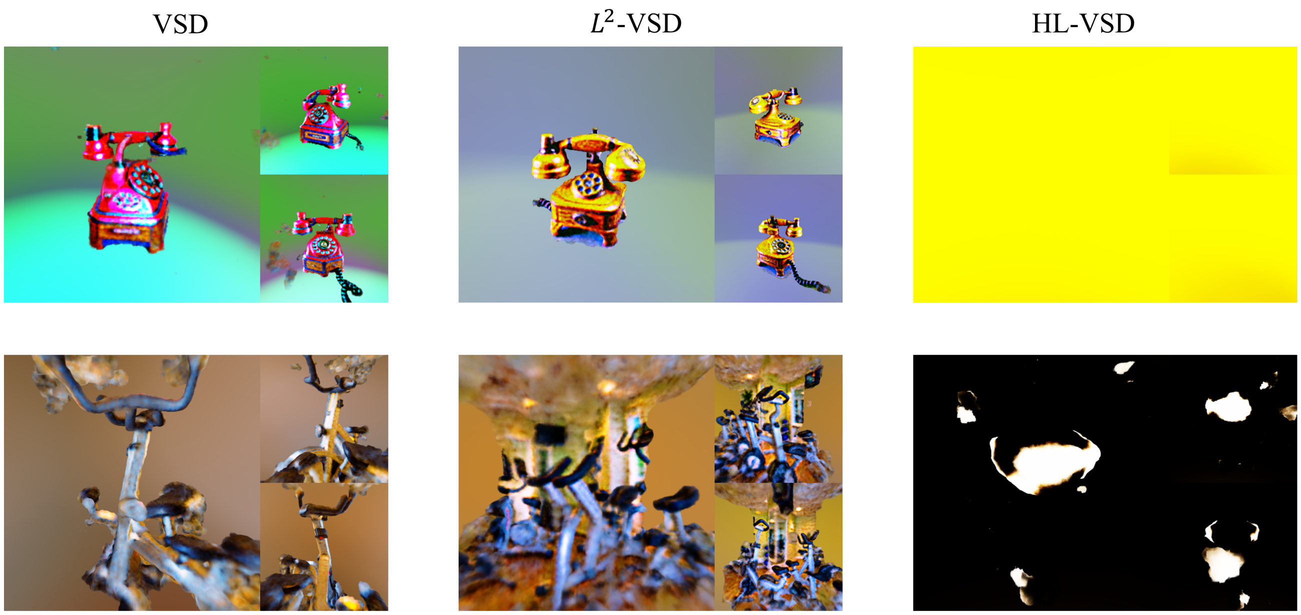


Figure 11. **Results comparison with using high-order term**. Prompts: (upper)"A rotary telephone carved out of wood" ;(Bottom)"a DSLR photo of an exercise bike in a well lit room"

D. Other Related works

D.1. Text-to-Image Diffusion Models

Text-to-image diffusion models [14, 15] are essential for text-to-3D generation. These models incorporate text embeddings during the iterative denoising process. Leveraging large-scale image-text paired datasets, they address text-to-image generation tasks. Latent diffusion models [16], which diffuse in low-resolution latent spaces, have gained popularity due to reduced computation costs. Additionally, text-to-image diffusion models find applications in various computer vision tasks, including text-to-3D [15, 22], image-to-3D [30], text-to-svg [5], and text-to-video [6, 21].

D.2. Text-to-3D Generation without 2D-Supervision

Text-to-3D generation techniques have evolved beyond relying solely on 2D supervision. Researchers explore diverse approaches to directly create 3D shapes from textual descriptions. Volumetric representations, such as 3D-GAN [26] and Occupancy Networks [10], use voxel grids [8, 25]. Point cloud generation methods, like PointFlow [32] and AtlasNet [28], work with sets of 3D points. Implicit surface representations, exemplified by DeepVoxels [23] and SIREN [24], learn implicit functions for shape surfaces. Additionally, graph-based approaches (GraphVAE [20], GraphRNN [33]) capture relationships between parts using graph neural networks.

D.3. Advancements in 3D Score Distillation Techniques

Various techniques enhance score distillation effectiveness. Magic3D [7] and Fantasia3D [1] disentangle geometry and texture optimization using mesh and DMTet [18]. TextMesh [27] and 3DFuse [17] employ depth-conditioned text-to-image diffusion priors for geometry-aware texturing. Score debiasing [3] and Perp-Neg [34] refine text prompts for better 3D generation. Researchers also explore timestep scheduling (DreamTime [4], RED-Diff [9]) and auxiliary losses (CLIP loss [31], adversarial loss [12]) to improve score distillation.

E. Discussion

Score Identity Distillation (SiD) [35] Apart from direct comparison with the text-to-3D score distillation method, our method can draw some similarities with some 2D diffusion distillation methods. SiD reformulates forward diffusion as semi-implicit distributions and leverages three score-related identities to create an innovative loss mechanism. The weighted loss is expressed as:

$$\begin{aligned}\tilde{\mathcal{L}}_{SiD}(\theta_i) = -\alpha\frac{w(t)}{\sigma_t^4}||\epsilon_{pretrain}(x_t,t) - \epsilon_\phi(x_t,t)||_2^2 \\ + \frac{w(t)}{\sigma_t^4}(\epsilon_{pretrain}(x_t,t) - \epsilon_\phi(x_t,t))^T(\epsilon_\phi(x_t,t) - \epsilon)\end{aligned} \tag{1}$$

where $x_t = g(\theta_i)$.Compared with the original VSD loss, the additional term in SiD has an important factor $(\epsilon_\phi - \epsilon)$, which corrects the original loss in a projected direction. This factor also exists in our term, so we assume that our first-order term shares some similarity with this correction term.

F. Gaussian Example Code

```python
import os
import math
import random
import numpy as np
from tqdm import tqdm, trange
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim.lr_scheduler import LambdaLR

def get_cosine_schedule_with_warmup(optimizer, num_warmup_steps, num_training_steps,
    min_lr=0., num_cycles: float = 0.5):

   def lr_lambda(current_step):
      if current_step < num_warmup_steps:
         return float(current_step) / float(max(1, num_warmup_steps))
      progress = float(current_step - num_warmup_steps) / float(max(1, num_training_steps -
          num_warmup_steps))
      return max(min_lr, 0.5 * (1.0 + math.cos(math.pi * float(num_cycles) * 2.0 *
          progress)))

   return LambdaLR(optimizer, lr_lambda, -1)

def seed_everything(seed):
   random.seed(seed)
   os.environ['PYTHONHASHSEED'] = str(seed)
   np.random.seed(seed)
   torch.manual_seed(seed)
   torch.cuda.manual_seed(seed)

```

```python
def sample_gassian(mu, sigma, N_samples=None, seed=None):
   assert N_samples is not None or seed is not None
   if seed is None:
      seed = torch.randn((N_samples, d), device=mu.device)
   samples = mu + torch.matmul(seed, sigma.t())
   return samples

# Core function: compute score function of perturbed Gaussian distribution
# \nabla \log p_t(x_t) = -(Simga^{-1} + sigma_t^2 I) (x_t - \alpha_t * \mu)
def calc_perturbed_gaussian_score(x, mu, sigma, alpha_noise, sigma_noise):
   if mu.ndim == 1:
      mu = mu[None, ...] # [d] -> [1, d]
   if sigma.ndim == 2:
      sigma = sigma[None, ...] # [d, d] -> [1, d, d]

   mu = mu * alpha_noise[..., None] # [B, d]
   sigma = torch.matmul(sigma, sigma.permute(0, 2, 1)) # [1, d, d]
   sigma = (alpha_noise**2)[..., None, None] * sigma # [B, d, d]
   sigma = sigma + (sigma_noise**2)[..., None, None] * torch.eye(sigma.shape[1], device=sigma.device)[None, ...] # [B, d, d]
   inv_sigma = torch.inverse(sigma) # [B, d, d]
   return torch.matmul(inv_sigma, (mu - x)[..., None]).squeeze(-1) # [B, d, d] @ [B, d, 1] -> [B, d, 1] -> [B, d]

# data dimension
N = 256
d = 2
ndim = d
lora_steps = 10
# set the hyperparameters
seed = 0
dist_0 = 10
lr = 1e-2
min_lr = 0
weight_decay = 0
warmup_steps = 100
total_steps = 2000
scheduler_type = 'cosine'
lambda_coeff = 1.0
method = 'l-vsd' # or 'real-vsd', 'vsd'
output_dir = ''
logging_steps = 10

device = torch.device('cuda:0')
seed_everything(seed)

# groundtruth distribution
p_mu = torch.rand(d, device=device) # uniform random in [0, 1] x [0, 1]
p_sigma = torch.rand((d, d), device=device) + torch.eye(d, device=device) # positive semi-definite

# diffusion coefficients
beta_start = 0.0001
beta_end = 0.02

# parametric distribution to optimize
q_mu = nn.Parameter(torch.rand(d, device=device) * dist_0 + p_mu)
q_sigma = nn.Parameter(torch.rand(d, d, device=device))
```

```
r_mu = nn.Parameter(torch.zeros(d, device=device)).to(device)
r_sigma = nn.Parameter(torch.zeros(d, d, device=device)).to(device)

optimizer = torch.optim.AdamW([q_mu, q_sigma], lr=lr, weight_decay=weight_decay)
scheduler = get_cosine_schedule_with_warmup(optimizer, warmup_steps, int(total_steps*1.5),
    min_lr) if scheduler_type == 'cosine' else None

# set the optimizer and scheduler of LoRA model
r_optimizer = torch.optim.AdamW([r_mu, r_sigma], lr=5*lr, weight_decay=weight_decay)

# saving checkpoints
state_dict = []
N_render = 4
# store per-step samples. fixed seed for visualization
vis_seed = torch.randn((1, N, d), device=device)
vis_seed_true = torch.randn((1, N, d), device=device)
vis_seed2 = torch.randn((1, N, d), device=device)
vis_samples = [] # [steps, p+q, N_samples, N_dim]
# x_previous = 0

for i in trange(total_steps + 1):
   optimizer.zero_grad()

   # sample time steps and compute noise coefficients
   betas_noise = torch.rand(N_render, device=device) * (beta_end - beta_start) + beta_start
   alphas_noise = torch.cumprod(1.0 - betas_noise, dim=0)
   sigmas_noise = ((1 - alphas_noise) / alphas_noise) ** 0.5

   # sample from g(x) = q_mu + q_sigma @ c, c ~ N(0, I)
   x = sample_gassian(q_mu, q_sigma, N_samples=N_render)
   # sample gaussian noise
   eps = torch.randn((N_render, d), device=device)
   # diffuse and perturb samples
   x_t = x * alphas_noise[..., None] + eps * sigmas_noise[..., None]

   # w(t) coefficients
   w = ((1 - alphas_noise) * sigmas_noise)[..., None]

   # compute score distillation update
   if method == 'l-vsd':
      xp = x.detach()
      for j in range(lora_steps):
         r_optimizer.zero_grad()
         q_muo = q_mu.detach()
         q_sigmao = q_sigma.detach()
         loss_r = F.mse_loss(q_muo, r_mu, reduction="sum") + F.mse_loss(q_sigmao, r_sigma,
             reduction="sum")

         loss_r.backward()
         r_optimizer.step()

   with torch.no_grad():
      # \nabla \log p_t(x_t)
      score_p = calc_perturbed_gaussian_score(x_t, p_mu, p_sigma, alphas_noise,
          sigmas_noise)

      if method == 'sds':
```

```
            # -[\nabla \log p_t(x_t) - eps]
            grad = -w * (score_p - eps)
        elif method == 'vsd':
            # \nabla \log q_t(x_t | c) - centering trick
            cond_mu = x.detach()
            cond_sigma = torch.zeros_like(q_sigma)
            score_q = calc_perturbed_gaussian_score(x_t, cond_mu, cond_sigma, alphas_noise,
                sigmas_noise)

            # -[\nabla \log p_t(x_t) - \nabla \log q_t(x_t | c)]
            grad = -w * (score_p - score_q)
        elif method == 'real-vsd' or method == 'l-vsd':
            cond_mu = r_mu.detach()
            cond_sigma = r_sigma.detach()
            score_q_appx = calc_perturbed_gaussian_score(x_t, cond_mu, cond_sigma,
                alphas_noise, sigmas_noise)

            grad = -w * (score_p - score_q_appx)

    # reparameterization trick for backpropagation
    # d(loss)/d(latents) = latents - target = latents - (latents - grad) = grad
    grad = torch.nan_to_num(grad)
    target = (x_t - grad).detach()
    loss = 0.5 * F.mse_loss(x_t, target, reduction="sum") / N_render

    loss.backward()
    optimizer.step()
    if scheduler is not None:
        scheduler.step()


    if method == 'real-vsd':
        r_mu_previous = r_mu.detach()
        r_sigma_previous = r_sigma.detach()
        xp = x.detach()
        for j in range(lora_steps):
            r_optimizer.zero_grad()
            q_muo = q_mu.detach()
            q_sigmao = q_sigma.detach()
            loss_r = F.mse_loss(q_muo, r_mu, reduction="sum") + F.mse_loss(q_sigmao, r_sigma,
                reduction="sum")

            loss_r.backward()
            r_optimizer.step()

    # logging
    if i % logging_steps == 0:
        state_dict.append({
            'step': i,
            'q_mu': q_mu.detach().cpu().numpy(),
            'q_sigma': q_sigma.detach().cpu().numpy(),
        })

        # save sample positions
        with torch.no_grad():
            p_samples = sample_gassian(p_mu, p_sigma, seed=vis_seed_true[0])
            p_samples = p_samples.detach().cpu().numpy()
```

```
        q_samples = sample_gassian(q_mu, q_sigma, seed=vis_seed[0])
        q_samples = q_samples.detach().cpu().numpy()

        if method == 'real-vsd':
            r_samples = sample_gassian(r_mu_previous, r_sigma_previous, seed=vis_seed2[0])
            r_samples = r_samples.detach().cpu().numpy()
        else:
            r_samples = sample_gassian(r_mu, r_sigma, seed=vis_seed2[0])
            r_samples = r_samples.detach().cpu().numpy()

        vis_samples.append(np.stack([p_samples, q_samples, r_samples], 0))
```

References

[1] Rui Chen, Yongwei Chen, Ningxin Jiao, and Kui Jia. Fantasia3d: Disentangling geometry and appearance for high-quality text-to-3d content creation, 2023. 8

[2] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 2

[3] Susung Hong, Donghoon Ahn, and Seungryong Kim. Debiasing scores and prompts of 2d diffusion for robust text-to-3d generation. *arXiv preprint arXiv:2303.15413*, 2023. 8

[4] Yukun Huang, Jianan Wang, Yukai Shi, Xianbiao Qi, Zheng-Jun Zha, and Lei Zhang. Dreamtime: An improved optimization strategy for text-to-3d content creation. *arXiv preprint arXiv:2306.12422*, 2023. 8

[5] Ajay Jain, Amber Xie, and Pieter Abbeel. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1911–1920, 2023. 7

[6] Levon Khachatryan, Andranik Movsisyan, Vahram Tadevosyan, Roberto Henschel, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. Text2video-zero: Text-to-image diffusion models are zero-shot video generators. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15954–15964, 2023. 7

[7] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation, 2023. 8

[8] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *Advances in neural information processing systems*, 32, 2019. 7

[9] Morteza Mardani, Jiaming Song, Jan Kautz, and Arash Vahdat. A variational perspective on solving inverse problems with diffusion models. *arXiv preprint arXiv:2305.04391*, 2023. 8

[10] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019. 7

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022. 2

[12] Tuomas Oikarinen, Wang Zhang, Alexandre Megretski, Luca Daniel, and Tsui-Wei Weng. Robust deep reinforcement learning through adversarial loss. *Advances in Neural Information Processing Systems*, 34:26156–26167, 2021. 8

[13] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 3

[14] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021. 7

[15] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022. 7

[16] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 7

[17] Junyoung Seo, Wooseok Jang, Min-Seop Kwak, Hyeonsu Kim, Jaehoon Ko, Junho Kim, Jin-Hwa Kim, Jiyoung Lee, and Seungryong Kim. Let 2d diffusion model know 3d-consistency for robust text-to-3d generation. *arXiv preprint arXiv:2303.07937*, 2023. 8

[18] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021. 8

[19] Yichun Shi, Peng Wang, Jianglong Ye, Long Mai, Kejie Li, and Xiao Yang. Mvdream: Multi-view diffusion for 3d generation. *arXiv:2308.16512*, 2023. 2

[20] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 412–422. Springer, 2018. 7

[21] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*, 2022. 7

[22] Uriel Singer, Shelly Sheynin, Adam Polyak, Oron Ashual, Iurii Makarov, Filippos Kokkinos, Naman Goyal, Andrea Vedaldi, Devi Parikh, Justin Johnson, et al. Text-to-4d dynamic scene generation. *arXiv preprint arXiv:2301.11280*, 2023. 7

[23] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019. 7

[24] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020. 7

[25] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 7

[26] Li Sun, Junxiang Chen, Yanwu Xu, Mingming Gong, Ke Yu, and Kayhan Batmanghelich. Hierarchical amortized training for memory-efficient high resolution 3d gan. *arXiv preprint arXiv:2008.01910*, 2020. 7

[27] Christina Tsalicoglou, Fabian Manhardt, Alessio Tonioni, Michael Niemeyer, and Federico Tombari. Textmesh: Generation of realistic 3d meshes from text prompts. *arXiv preprint arXiv:2304.12439*, 2023. 8

[28] Maria Vakalopoulou, Guillaume Chassagnon, Norbert Bus, Rafael Marini, Evangelia I Zacharaki, M-P Revel, and Nikos Paragios. Atlasnet: Multi-atlas non-linear deep networks for medical image segmentation. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2018: 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part IV 11*, pages 658–666. Springer, 2018. 7

[29] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[30] Dejia Xu, Yifan Jiang, Peihao Wang, Zhiwen Fan, Yi Wang, and Zhangyang Wang. Neurallift-360: Lifting an in-the-wild 2d photo to a 3d object with 360deg views. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4479–4489, 2023. 7

[31] Dejia Xu, Yifan Jiang, Peihao Wang, Zhiwen Fan, Yi Wang, and Zhangyang Wang. Neurallift-360: Lifting an in-the-wild 2d photo to a 3d object with 360deg views. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4479–4489, 2023. 8

[32] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019. 7

[33] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep autoregressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018. 7

[34] Minda Zhao, Chaoyi Zhao, Xinyue Liang, Lincheng Li, Zeng Zhao, Zhipeng Hu, Changjie Fan, and Xin Yu. Efficientdreamer: High-fidelity and robust 3d creation via orthogonal-view diffusion prior. *arXiv preprint arXiv:2308.13223*, 2023. 8

[35] Mingyuan Zhou, Huangjie Zheng, Zhendong Wang, Mingzhang Yin, and Hai Huang. Score identity distillation: Exponentially fast distillation of pretrained diffusion models for one-step generation. *arXiv preprint arXiv:2404.04057*, 2024. 8